Express Mail Label No. EL692235071US

DECLARATION AND POWER OF ATTORNEY FOR UTILITY OR DESIGN PATENT APPLICATION		
☒ Declaration Submitted with Initial Filing	☐ Declaration Submitted after Initial Filing (surcharge 37 CFR 1.16(e) required)	
	Attorney Docket No.	SRI-009B (7565/13)
	First Named Inventor	Ogier
	COMPLETE IF KNOWN	
	Application Serial Number	Not Yet Assigned
	Filing Date	Herewith
	Group Art Unit	Not Yet Assigned
	Examiner Name	Not Yet Assigned

As a below named inventor, I hereby declare that:

My residence, post office address, and citizenship are as stated below next to my name.

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled:

A System and Method for Disseminating Topology and Link-State Information to Routing Nodes in a Mobile Ad Hoc Network

(Title of the Invention)

the specification of which

☒ is attached hereto

OR

☐ was filed on (MM/DD/YYYY) _____ as United States Application Serial Number or PCT International Application Number _____ and was amended on (MM/DD/YYYY) _____ *(if applicable).*

I hereby state that I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment specifically referred to above.

I acknowledge the duty to disclose to the Patent Office all information known by me to be material to patentability as defined in 37 CFR 1.56.

I hereby claim foreign priority benefits under 35 U.S.C. 119(a)-(d) or 365(b) of any foreign application(s) for patent or inventor's certificate, or 365(a) of any PCT international application which designated at least one country other than the United States of America, listed below and have also identified below, by checking the box, any foreign application for patent or inventor's certificate, or of any PCT international application having a filing date before that of the application on which priority is claimed.

Prior Foreign Application Number(s)	Country	Foreign Filing Date (MM/DD/YYYY)	Priority Not Claimed	Certified Copy Attached? YES	NO
			☐	☐	☐
			☐	☐	☐
			☐	☐	☐
			☐	☐	☐

☐ Additional foreign application numbers are listed on a supplemental priority data sheet attached hereto.

I hereby claim the benefit under 35 U.S.C. 119(e) of any United States provisional application(s) listed below.

Application Serial Number(s)	Filing Date (MM/DD/YYYY)	
60/232,047	09/12/2000	☐ Additional provisional application serial numbers are listed on a supplemental priority data sheet attached hereto.
60/_____	11/14/2000	

DECLARATION – Utility or Design Patent Application

I hereby claim the benefit under 35 U.S.C. 120 of any United States application(s), or 365(c), of any PCT international application designating the United States of America, listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States or PCT International application in the manner provided by the first paragraph of 35 U.S.C. 112, I acknowledge the duty to disclose information which is material to patentability as defined in 37 CFR 1.56 which became available between the filing date of the prior application and the national or PCT international filing date of this application.

U.S. Parent Application or PCT Parent Serial Number	Parent Filing Date (MM/DD/YYYY)	Parent Patent Number *(if applicable)*

☐ Additional U.S. or PCT international application numbers are listed on a supplemental priority data sheet attached hereto.

As a named inventor, I hereby appoint the following registered practitioners to prosecute this application and to transact all business in the Patent and Trademark Office connected therewith: ☐ Customer Number

OR

☒ Registered practitioner(s) name/registration number listed below

→ *Place Customer Number Bar Code Label Here*

Name	Registration Number	Name	Registration Number
Steven M. Bauer	31,481	Thomas C. Meyers	36,989
John V. Bianco	36,748	Joseph B. Milstein	42,897
Isabelle A.S. Blundell	43,321	David G. Miranda	42,898
Maureen A. Bresnahan	44,559	Ronda P. Moore	44,244
Michael H. Brodowski	41,640	Indranil Mukerji	P-46,944
Jennifer A. Camacho	43,526	Edmund R. Pitcher	27,829
Joseph A. Capraro, Jr.	36,471	Michael A. Rodriguez	41,274
John J. Cotter	38,116	Jamie H. Rose	45,054
John V. Forcier	42,545	R. Stephen Rosenholm	45,283
Steven J. Frank	33,497	Christopher W. Stamos	35,370
Brian M. Gaff	44,691	Diana M. Steel	43,153
Michael J. Giannetta	42,574	Joseph P. Sullivan	45,349
Duncan A. Greenhalgh	38,678	Robert J. Tosti	35,393
William G. Guerin	41,047	Thomas A. Turano	35,722
Jonathan A. Harris	44,744	Michael J. Twomey	38,349
Ira V. Heffan	41,059	Christine C. Vito	39,061
Danielle L. Herritt	43,670	Patrick R.H. Waller	41,418
Douglas J. Kline	35,574	Daniel A. Wilson	45,508
John D. Lanza	40,060	Yin P. Zhang	44,372
Kurt W. Lockwood	40,704		

☐ Additional registered practitioners named on supplemental Registered Practitioner Information sheet attached hereto.

Direct all correspondence to:

Patent Administrator
Testa, Hurwitz & Thibeault, LLP
High Street Tower
125 High Street
Boston, MA 02110
Tel. No.: (617) 248-7000
Fax No.: (617) 248-7100

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under 18 U.S.C. 1001 and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Name of Sole or First Inventor:	☐ A petition has been filed for this unsigned inventor							
Given Name (first and middle [if any])			Family Name or Surname					
Richard G.			**Ogier**					
Inventor's Signature	Richard G. Ogier					Date	11/30/2000	
Residence	City	**Half Moon Bay**	State	**CA**	Country	**USA**	Citizenship	**USA**
Post Office Address	**585C Kelly Street**							
P.O. Address (line 2)	City	**Half Moon Bay**	State	**CA**	ZIP	**94019**	Country	**USA**
☐ Additional inventors are being named on the ____ supplemental Additional Inventor(s) sheet(s) attached hereto.								
Name of Additional Joint Inventor, if any:	☐ A petition has been filed for this unsigned inventor							
Given Name (first and middle [if any])			Family Name or Surname					
Bhargav R.			**Bellur**					
Inventor's Signature	B. R. Bhargav					Date	11/30/2000	
Residence	City	**Fremont**	State	**CA**	Country	**USA**	Citizenship	**India**
Post Office Address	**5135 Shalimar Circle**							
P.O. Address (line 2)	City	**Fremont**	State	**CA**	ZIP	**94019**	Country	**USA**
Name of Additional Joint Inventor, if any:	☐ A petition has been filed for this unsigned inventor							
Given Name (first and middle [if any])			Family Name or Surname					
Fred Lambert			**Templin**					
Inventor's Signature	Fred L. Templin					Date	11/30/2000	
Residence	City	**Portola Valley**	State	**CA**	Country	**USA**	Citizenship	**USA**
Post Office Address	**291 La Cuesta Drive**							
P.O. Address (line 2)	City	**Portola Valley**	State	**CA**	ZIP	**94028**	Country	**USA**

APPENDIX A

Network-Level Procedures

The notation LSU(update_list) represents a link-state-update message that includes the updates (u, v, c, sn) in the update_list.

```
Process_Update(i, nbr, in_message){
    // Called when an update message in_message is received from nbr.
    Update_Topology_Table(i, nbr, in_message, update_list).
    Update_Parents(i).
    For each node src in TT_i {
        Let update_list(src) consist of all tuples (k, l, c, sn) in update_list such that
        k = src.
        If update_list(src) is nonempty
            Send message LSU(update_list(src)) to children_i(src).}}

Update_Topology_Table(i, nbr, in_message, update_list){
    Set update_list to empty list.
    For each ((u,v,c,sn) in in_message) {
        If (p_i(u) == nbr) {
            If ((u,v) is in TT_i and sn > TT_i(u,v).sn) {
                Add (u,v,c,sn) to update_list.
                Set TT_i(u,v).sn = sn.
                Set TT_i(u,v).c = c.
                If (sn > sn_i(u)) Set sn_i(u) = sn.}
            If ((u,v) is not in TT_i) {
                Add (u,v,c,sn) to TT_i.
                Add (u,v,c,sn) to update_list.
                If (sn > sn_i(u)) Set sn_i(u) = sn.}}}}

Link_Change(i,j){
    // Called when the cost of link (i,j) changes.
    If (|TT_i(i,j).c - cost(i,j)|/TT_i(i,j).c > epsilon) {
        Set TT_i(i,j).c = cost(i,j).
        Set TT_i(i,j).sn = current time stamp SN_i.
        Set update_list = {(i, j, TT_i(i, j).c, TT_i(i, j).sn)
        Send message LSU(update_list) to children_i(i).}}

Link_Down(i,j){
    // Called when link (i,j) goes down.
    Remove j from N_i.
    Set TT_i(i,j).c = infinity.
```

```
        Set TT_i(i,j).sn = current time stamp SN_i.
        Update_Parents(i).
        For each (node src in TT_i) remove j from children_i(src).
        Set update_list = {(i,j, infinity, TT_i(i,j).sn)}.
        Send message LSU(update_list) to children_i(i).}

    Link_Up(i,j){
        // Called when link (i,j) comes up.
        Add j to N_i.
        Set TT_i(i,j).c = cost(i,j).
        Set TT_i(i,j).sn = current time stamp SN_i.
        Update_Parents(i).
        Set update_list = {(i, j, TT_i(i,j).c, TT_i(i,j).sn)}.
        Send message LSU(update_list) to children_i(i).}

    Update_Parents(i){
        Compute_New_Parents(i)
        For each (node k in N_i){
            Set cancel_src_list(k), src_list(k), and sn_list(k) to empty.}
        For each (node src in TT_i such that src != i){
            If (new_p_i(src) != p_i(src)){
                If (p_i(src) != NULL){
                    Set k = p_i(src).
                    Add src to cancel_src_list(k).}
                Set p_i(src) = new_p_i(src).
                If (new_p_i(src) != NULL){
                    Set k = new_p_i(src).
                    Add src to src_list(k).
                    Add sn_i(src) to sn_list(k).}}}
        For each (node k in N_i){
            If (src_list(k) is nonempty){
                Send message NEW PARENT(src_list(k), sn_list(k)) to k.}
            If (cancel_src_list(k) is nonempty{
                Send message CANCEL PARENT(cancel_src_list(k)) to k.}}}

    Compute_New_Parents(i){
        For each (node src in TT_i such that src != i){
            Set new_p_i(src) = NULL.}
        Compute min-hop paths using Dijkstra.
        For each (node src in TT_i such that src != i){
            Set new_p_i(src) equal to the neighbor of node i along the minimum-hop
            path from i to src.}}

    Process_New_Parent(i, nbr, src_list, sn_list){
        // Called when node i receives a NEW PARENT(src_list, sn_list) message from
        nbr.
        Set update_list to empty list.
```

```
        For each (node src in src_list) {
            Let sn_list.src denote the sequence number corresponding to src in sn_list.
            Add nbr to children_i(src).
            Set new_updates = {(k, l, c, sn) in TT_i such that k = src and sn >
            sn_list.src}.
            Add new_updates to update_list.}
        Send message LSU(update_list) to nbr.}

    Process_Cancel_Parent(i,nbr,src_list ){
        // Called when node i receives a CANCEL PARENT(src_list) message from nbr.
        For each (node src in src_list) remove nbr from children_i(src).}

    Send_Periodic_Updates(i){
        Set update_list to empty.
        For each (j in N_i such that TT_i(i,j). c != infinity){
            Set TT_i(i,j).sn = current time stamp SN_i.
            Add (i, j, TT_i(i,j).c, TT_i(i,j).sn) to update_list. }
        Send message LSU(update_list) to children_i(i).}

    Compute_New_Parents2(i){
        S ← ∅;
        For each (v ∈ TT_i) {
            Set d(v) = infinity;
            Set pred(v) = NULL;
            Set new_p_i(v) = NULL; }
        d(i) ← 0;
        While (there exists w ∈ TT_i – S such that d(w) < infinity){
            Set u = node w ∈ TT_i – S that minimizes d(w);
            Set S = S ∪ {u};
            For each (v such that (u, v) ∈ TT_i) {
                If (d(u) + 1 < d(v) or [d(u) + 1 = d(v) and new_p_i(u) = p_i(v)]) {
                    Set d(v) = d(u) + 1;
                    Set pred(v) = u;
                    If (u = i)  Set new_p_i(v) = v;
                    Else Set new_p_i(v) = new_p_i(u); }}}}
```

Partial-Topology 1

The function Mark_Special_Links() is called whenever the parent p_i(src) or the set of children children_i(src) for any source src changes. The notation LSU(update_list) represents a link-state-update message that includes the updates (u, v, c, sn, sp) in the update_list, where sp is

a single bit that indicates whether the link is "special", i.e., whether it should be broadcast to all nodes.

```
Mark_Special_Links(i){
    For all (outgoing links (i,j)) {Set TT_i(i,j).sp = 0;}
    For all (nodes src != i){
        if (p_i(src) != NULL and p_i(src) != src){
            Set TT_i(i, p_i(src)).sp = 1;}   //Link is special.
            For all (nodes j in children_i(src)){
                Set TT_i(i,j).sp = 1;}   //Link is special.
    }
}

Update_Topology_Table(i, nbr, in_message, update_list){
    Set update_list to empty list.
    For each ((u,v,c,sn,sp) in in_message) {
        If (p_i(u) = nbr) {
            If ((u,v) is in TT_i and sn > TT_i(u,v).sn) {
                Set TT_i(u,v).sn = sn.
                Set TT_i(u,v).c = c.
                Set TT_i(u,v).sp = sp.
                (Only links marked as special are forwarded.)
                If (sp = 1) Add (u,v,c,sn,sp) to update_list.
                If (sn > sn_i(u)) Set sn_i(u) = sn.}
            If ((u,v) is not in TT_i) {
                Add (u,v,c,sn,sp) to TT_i.
                If (sp = 1) Add (u,v,c,sn,sp) to update_list.
                If (sn > sn_i(u)) Set sn_i(u) = sn.}}}}

Process_Update(i, nbr, in_message){
    // Called when an update message in_message is received from nbr.
    Update_Topology_Table(i, nbr, in_message, update_list).
    Update_Parents(i).
    Mark_Special_Links(i).
    For each node src in TT_i {
        Let update_list(src) consist of all tuples (k, l, c, sn, sp) in update_list such
        that k = src.
        If update_list(src) is nonempty
            Send message LSU(update_list(src)) to children_i(src).}}

Link_Change(i,j){
    // Called when the cost of link (i,j) changes.
    If (|TT_i(i,j).c - cost(i,j)|/TT_i(i,j).c > epsilon) {
        Set TT_i(i,j).c = cost(i,j).
        Set TT_i(i,j).sn = current time stamp SN_i.
```

```
                Set update_list = {(i, j, TT_i(i, j).c, TT_i(i, j).sn, TT_i(i,j).sp)}.
                Send message LSU(update_list) to children_i(i).}}

        Link_Down(i,j){
                // Called when link (i,j) goes down.
                Remove j from N_i.
                Set TT_i(i,j).c = infinity.
                Set TT_i(i,j).sn = current time stamp SN_i.
                Update_Parents(i).
                For each (node src in TT_i) remove j from children_i(src).
                Mark_Special_Links(i).
                Set update_list = {(i,j, infinity, TT_i(i,j).sn, TT_i(i,j).sp)}.
                Send message LSU(update_list) to children_i(i).}

        Link_Up(i,j){
                // Called when link (i,j) comes up.
                Add j to N_i.
                Set TT_i(i,j).c = cost(i,j).
                Set TT_i(i,j).sn = current time stamp SN_i.
                Update_Parents(i).
                Mark_Special_Links(i).
                Set update_list = {(i, j, TT_i(i,j).c, TT_i(i,j).sn, TT_i(i,j).sp)}.
                Send message LSU(update_list) to children_i(i).}

        Update_Parents(i){
                Compute_New_Parents(i).
                For each (node k in N_i)
                        Set cancel_src_list(k), src_list(k), and sn_list(k) to empty.
                For each (node src in TT_i such that src != i){
                        If (new_p_i(src) != p_i(src)){
                                If (p_i(src) != NULL){
                                        Set k = p_i(src).
                                        Add src to cancel_src_list(k).}
                                Set p_i(src) = new_p_i(src).
                                If (new_p_i(src) != NULL){
                                        Set k = new_p_i(src).
                                        Add src to src_list(k).
                                        Add sn_i(src) to sn_list(k).}}}
                For each (node k in N_i){
                        If (src_list(k) is nonempty){
                                Send message NEW PARENT(src_list(k), sn_list(k)) to k.}
                        If (cancel_src_list(k) is nonempty{
                                Send message CANCEL PARENT(cancel_src_list(k)) to k.}}}

        Compute_New_Parents(i){
                For each (node src in TT_i such that src != i){
                        Set new_p_i(src) = NULL.}
```

```
        Compute min-hop paths using Dijkstra.
        For each (node src in TT_i such that src != i){
            Set new_p_i(src) equal to the neighbor of node i along the minimum-hop
            path from i to src.}}

    Process_New_Parent(i, nbr, src_list, sn_list){
        //Called when node i receives a NEW PARENT(src_list, sn_list) message from
        nbr.
        Set update_list to empty list.
        For each (node src in src_list) {
            Let sn_list.src denote the sequence number corresponding to src in sn_list.
            Add nbr to children_i(src).
            If (src != i) Set TT_i(i, nbr).sp = 1.  //Link to nbr is special.
            If (src = i)  Set new_updates = {(src, v, c, sn, sp) in TT_i such that
                    sn > sn_list.src}.
            If (src != i) Set new_updates = {(src, v, c, sn, sp) in TT_i such that
                    sn > sn_list.src and sp = 1}.  //Only special links are sent.
            Add new_updates to update_list.}
        Send message LSU(update_list) to nbr.}

    Process_Cancel_Parent(i,nbr,src_list ){
        // Called when node i receives a CANCEL PARENT(src_list) message from nbr.
        For each (node src in src_list) remove nbr from children_i(src).
        Mark_Special_Links(i). }

    Send_Periodic_Updates(i){
        Set update_list to empty.
        For each (j in N_i such that TT_i(i,j).c != infinity){
            Set TT_i(i,j).sn = current time stamp SN_i.
            Add (i, j, TT_i(i,j).c, TT_i(i,j).sn, TT_i(i,j).sp) to update_list. }
        Send message LSU(update_list) to children_i(i).}
```

Partial-Topology 2

```
    Update(i, k, in_message){
        Update_Topology Table(i, k, in_message);
        Lex_Dijkstra;  // Uses lexicographic Dijkstra to compute Ti
        Generate_Updates(i, update_list);
        if (k does not equal i and update_list is non-empty){
            Send_Updates_Children(i, update_list);
        Update_Parents(i);
    }

    Send_Updates_Children(i, update_list){
        For each (node k ∈ Ni) {out_message(k) ← 0;}
        For each (node src ∈ TT_i s.t. src does not equal i){
            update_list(src) ← {(k, l, c) ∈ update_list s.t. k = src};
```

```
                for each (node k ∈ children_i(src)){
                    Add update_list(src) to out_message(k);}
            }
            For each (node k ∈ Ni s.t. out_message(k) is non-empty){
                Send the message out_message(k) to node k;}
        }

        Update_Topology_Table(i, k, in_message){
            For each ((u, v, c) ∈ in_message{
                // Process only updates received from the parent p_i(u)
                if (p_i(u) = k or k = i){
                    if ((u, v) ∉ TT_i or c! = TT_i(u, v).c{
                        TT_i(u, v) ← (u, v, c);
                        Mark (u, v) as changed in TT_i;}
                }
            }
            if (in_message is a PARENT_RESPONSE){
                For each (u such that in_message includes source u){
                    if (p_i(u) = k and pending_i(u) = 1){
                        pending_i(u) = 0;
                        For each (v such that TT_i contains an entry for (u, v)){
                            if (in_message does not contain update for link (u,
                            v)){
                                TT_i(u, v).c ←∞;
                                // indicates link should be deleted
                                Mark (u, v) as changed in TT_i;
                            }
                        }
                    }
                }
            }
        }

        Process_Cancel_Parent(i, nbr, src_list){
            For each (src ∈ src_list)
                children_i(src) ← children_i(src) – {nbr};
        }

        Generate_Updates(i, update_list){
            update_list ← 0;
            for each (entry (u, v, c, c') ∈ TT_i){
                if ((u, v) is in new Ti and ((u, v) is marked as changed or is not in old
                Ti)){
                    Add (u, v, c) to update_list;
                    Ti(u, v).c' ← Ti(u, v).c;
                    Ri ← Ri ∪{(u, v)};
```

```
                    }
                    else if ((u, v) is in Ri but not in new Ti and c > c'){
                            Add (u, v, ∞) to update_list;  // delete update
                            Ti(u, v).c' ← ∞;
                            Remove (u, v) from Ri;
                    }
                    if (TT_i(u, v).c = ∞)
                            Remove (u, v) from TT_i;
            }
    }

    Update_Parents(i){
            For each (node k ∈ Ni){
                    cancel_src_list(k) ← 0;
                    src_list(k) ← 0;}
            For each (node src ∈TT_i such that src ≠ i) {
                    new_p_i(src) ← next node on shortest path to src;
                    if (new_p_i(src) ≠p_i(src)){
                            if (new_p_i(src) ≠ NULL) {
                                    k ← p_i(src);
                                    cancel_src_list(k) ← cancel_src_list(k) ∪ {src};
                            }
                            if (new_p_i(src) ≠ NULL){
                                    k ← new_p_i(src);
                                    src_list(k) ← src_list(k) ∪ {src};
                            }
                            p_i(src) ← new_p_i(src);
                    }
            }
            For each (node k ∈ Ni){
                    if (src_list(k) ≠ 0)
                            Send NEW_PARENT(src_list(k)) to node k;
                    if(cancel_src_list(k) ≠ 0)
                            Send CANCEL_PARENT(cancel_src_list(k)) to node k;
            }
    }

    Process_New_Parent(i, nbr, src_list){
            update_list ← 0;
            for each (node u ∈ u_list) {
                    children_i(u) ← children_i(u) ∪ {nbr};
                    updates(u) ← {(u, v, c) ∈ TT_i such that (u, v) ∈ Ti};
                    update_list ← update_list ∪ updates (u);
            }
            Send PARENT RESPONSE(src_list, update_list) to nbr;}
```

Express Mail Label No. EL692235071US

STATEMENT CLAIMING SMALL ENTITY STATUS
(37 CFR 1.9(f) & 1.27(d))--NONPROFIT ORGANIZATION

Applicant: SRI INTERNATIONAL
Application No.: submitted herewith
Filed: SEPTEMBER 12, 2000
Title: TECHNIQUES FOR IMPROVED TOPOLOGY BROADCAST BASED ON REVERSE-PATH FORWARDING

I hereby state that I am an official empowered to act on behalf of the nonprofit organization identified below:
NAME OF NONPROFIT ORGANIZATION: SRI INTERNATIONAL
ADDRESS OF NONPROFIT ORGANIZATION: 333 Ravenswood Avenue
Menlo Park, CA 94025

TYPE OF NONPROFIT ORGANIZATION: UNIVERSITY OR OTHER INSTITUTION OF HIGHER EDUCATION
TAX EXEMPT UNDER INTERNAL REVENUE SERVICE CODE (26 U.S.C. 501(a) and 501(c)(3))

NONPROFIT SCIENTIFIC OR EDUCATIONAL UNDER STATUTE OF STATE OF THE UNITED STATES OF AMERICA
NAME OF STATE *CALIFORNIA*
CITATION OF STATUTE California Corporations Code Section 5110 et seq.

WOULD QUALIFY AS TAX EXEMPT UNDER INTERNAL REVENUE SERVICE CODE (26 U.S.C. 501(a) and 501(c)(3)) IF LOCATED IN THE UNITED STATES OF AMERICA
WOULD QUALIFY AS NONPROFIT SCIENTIFIC OR EDUCATIONAL UNDER STATUTE OF STATE OF THE UNITED STATES OF AMERICA IF LOCATED IN THE UNITED STATES OF AMERICA
(NAME OF STATE ________________)
(CITATION OF STATUTE ________________)

I hereby state that the nonprofit organization identified above qualifies as a nonprofit organization as defined in 37 CFR 1.9(e) for purposes of paying reduced fees to the United States Patent and Trademark Office regarding the invention described

[ ] in:
[XX] the specification filed herewith with title as listed above.
[ ] the application identified above.
[ ] the patent identified above.

I hereby state that rights under contract or law have been conveyed to and remain with the nonprofit organization regarding the above identified invention. If the rights held by the nonprofit organization are not exclusive, each individual, concern, or organization having rights in the invention must file separate statements as to their status as small entities and that no rights to the invention are held by any person, other than the inventor, who would not qualify as an independent inventor under 37CFR 1.9(c) if that person made the invention, or by any concern which would not qualify as a small business concern under 37 CFR 1.9(d) or a nonprofit organization under 37 CFR 1.9(e).

[ ] Each person, concern, or organization having any rights in the invention is listed below:
[XX] no such person, concern, or organization exists.
[ ] each such person, concern, or organization is listed below.

I acknowledge the duty to file, in this application or patent, notification of any change in status resulting in loss of entitlement to small entity status prior to paying, or at the time of paying, the earliest of the issue fee or any maintenance fee due after the date on which status as a small entity is no longer appropriate. (37 CFR 1.28(b))

NAME OF PERSON SIGNING: Richard Cramer
TITLE IN ORGANIZATION OF PERSON SIGNING Assistant Secretary
ADDRESS OF PERSON SIGNING: SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025

SIGNATURE ________________ DATE September 12, 2000